git log - command that lets you see all of the commits that have been created

- unique ID for each commit based on time and content of commit
- q to get out of git log

Local repository and online repository

git status - command that lets you view the status of change

git diff - will show you exactly what the difference is between what was in the repo and what changes we are introducing

git status and git diff are used on a regular basis. want to be aware of what you commit.

naming commits

should be able to go back and look at a repo's commits and understand what happened for each commit

each commit should be signed as a logical step, can't just name it "new code"

commits should be large enough to contain meaningful changes but small enough to reduce risk of losing progress

git commit -m "fixing a typo in the hyperlink"

Your branch is ahead of 'origin/master' by 1 commit means that on our laptop, we are one commit ahead of what is on Github, and we haven't pushed it yet, only committed locally

git push origin master

git remote -v

fetch takes info from Github and puts it onto laptop, push takes info from laptop and puts it onto Github

undoing changes

We want to preserve both versions of history but we also want a way to overwrite history if we want to

git rebase -i, will open in vim, type :wq to quit, removes commit locally, doing git push origin master at this point will result in a conflict between remote and local

we need to tell it that we want to overwrite history, so git push origin master -f

fetch and rebase

assume another user made a commit to the repo that we don't have locally

we need to fetch the changes

use git log, git status to see current status

git fetch origin will get the references of the changes but will not apply them, it will see that we are behind by one commit

we can do git pull to grab changes, but that is not good practice because it will try to merge and resolve conflicts

git rebase origin master is the better way

now, git log will reflect all the commits

don't delete folder and reclone repo, fetch changes and rebase

collaboration

one true version that no one can edit, each person makes their own copy and edits them on their own, and then requests to merge their version with the true version, this request to merge code is done by a pull request

make a fork of the main repo which will create a copy for you that you can make all your changes to

the main repository will have changed since the time you forked the repo, so it's your responsibility to keep the two branches in sync.

- you would need to take the new commits made on the main branch, fetch them, and merge them onto your local repo

should never commit directly to the master branch, any time you want to add anything, you create a new branch

- master branch should always work and have no conflicts

upstream is the content that you're copying from, origin is the copy that you own locally

- you create a new feature branch locally, then you can easily rebase the some-feat branch, then make a new pull request and it gets merged

how to code

software gets replaced because it becomes too complicated and no one can contribute anything to it in a way that's safe and has guarantees and runs and passes tests etc.

how to build code properly - knowing english words and grammar does not make you a poet, there's a couple good ways of coding things properly

time it takes to code something is an important factor too - spending too long to code something is wasting time

need to look at structure of the code, how is someone going to use this code? minimize side effects.

do one thing - each piece of code you write should only do a single thing. make clear blocks and components that you can later combine so that you know where your errors are coming from. reduced space to look for bug means you can find bugs faster.

two ways to code - top down approach and bottom up approach
top down - implement the helper functions, only implement what you need
bottom up - try to foresee all the components you are going to need, then combine them, keep going until you reach the top of the tree, which gets you the final function at the highest level that you were supposed to implement, nice because you have runnable code at every step, but you might end up implementing something you don't need because it's hard to foresee everything you need

boring code is good code, easy to read, simple, understand exactly what's happening without having to run the code

start vague and refine, don't commit to specificity too soon, what is the superset of the problem you are trying to solve?

many functions with small bodies is better than few functions with large bodies because you can use and reuse the smaller components